Tribhuvan University
Institute of Science and Technology
BSc. CSIT, Sixth Semester

Course Title: Complier Design and Construction
Course no: CSC-352 **Full Marks**: 60+20+20
Credit hours: 3 **Pass Marks**: 24+8+8

Nature of course: Theory (3 Hrs.) + Lab (3 Hrs.)

Course Synopsis: Analysis of source program. The phases of compiler.

Goal: This course introduces fundamental concept of compiler and its different phases.

Course Contents:

Unit 1:

1.1 Introduction to compiling: Compilers, Analysis of source program, the phases of compiler, compiler-construction tools. **4 Hrs.**

1.2 A Simple One-Pass Compiler: Syntax Definition, Syntax directed translation, Parsing, Translation for simple expression, Symbol Table, Abstract Stack Machines. **5 Hrs.**

Unit 2:

2.1 Lexical Analysis: The role of the lexical analyzer, Input buffering, Specification and recognition of tokens, Finite Automata, conversion Regular Expression to an NFA – Thompson's Construction, NFA to DFA – Subset Construction, regular expression to DFA, State minimization in DFA, Flex/Lex introduction **8 Hrs.**

2.2 Syntax Analysis: The role of parser, Context free grammar, Writing a grammar, Top-down parsing – recursive decent parsing, non-recursive predictive parsing, error recovery mechanism, LL grammar, Bottom-up parsing – handles, shift reduced parsing, LR parsers – SLR, LALR, LR, LR/LALR Grammars, Parser Generator **10 Hrs.**

Unit 3:

3.1 Syntax directed translation: Syntax directed definitions, syntax tree construction, synthesized and inherited attributes, dependency graph, S-attributed definitions, L-attributed definition, and Translation schemes, top-down and bottom-up evaluation. **5 Hrs.**

3.2 Type Checking: Type system, Specification simple type checker, equivalence of type expression, Type conversion, Type checking Yacc/Bison **3 Hrs.**

Unit 4:

4.1 Intermediate languages, three address code, Declarations, assignment statement, addressing array elements, Boolean expression, case statements, procedure calls, back patching **4 Hrs.**

4.2 Code generation and Optimization: code generator design issues, target machine, runtime storage management, basic blocks and flow graphs, next use information, simple code generator, Peephole optimization **6 Hrs.**

Laboratory works:

1. Writing a complier, optimization techniques, comparing the compilers.
2. Construction of Lexical Analyzer.
3. Construction of Parser
4. Development of Code Generator
5. Write a code to show the function of symbol table.
6. Implement the parsing techniques.
7. Show the application of different types of grammar.
8. Implement the lexical analyzer generator.
9. Implement the type conversation.
10. The course instructor is allowed to create a group two students.
    a. Assign them to write a small compiler.

Text Books: Compilers, Principles, Techniques, and Tools, Pearson education Asia.
Reference:

Homework
Assignment: Assignment should be given from the above units in throughout the semester.

Computer Usage: No specific

Prerequisite: C, C++, Data Structure, Automata Theory

Category Content: Science Aspect: 25%
Design Aspect: 75%

Tribhuvan University
Institute of Science and Technology
Bachelor of Computer Science and Information Technology
Course Title: Compiler Design and Construction
Model Question Paper

Course No.: CSC-352 **Full Marks: 60**

Pass Marks: 24 **Time: 3 hours.**

(There may be 10 questions each of carrying 6 marks or 5 questions with partitions each of carrying 12 marks in total)

Attempt all questions. [10x6=60]

1. Discuss the phases of compiler construction briefly.

2. Discuss the role of symbol table in compiler design.

3. Why regular expressions are used in token specification? Write the regular expression to specify the identifier like in C.

4. Consider the grammar:
$$E \rightarrow TE'$$
$$E' \rightarrow +TE'|\varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT'|\varepsilon$$
$$F \rightarrow (E)\ |id$$
Compute the FIRST and FOLLOW for each symbol.

5. Discuss with a suitable example the operation of stack implementation of shift-reduce parsing.

6. Define the L-attributed definitions. How L-attributed definitions are evaluated?

7. Define the process for Bottom-Up Evaluation of Inherited Attributes.

8. Consider the grammar:
$$E \rightarrow E+T|T$$
$$T \rightarrow num.num|num$$
The grammar generates the expression of +to integer or real. Give a syntax-directed definition to determine the type of expression. When two integers are added, the resulting type is integer otherwise, it is real.

9. Write the grammar with semantic rules that translate the C like ***while statement*** into three addresses code representation.

10. How next-use information is useful in code generation? Explain steps of computing next-use information.